Exercise 1

Write a C program able to read a list of integer numbers from a text file (whose name is given as a command line parameter) printing on another text file (whose name is given as a second command line parameter) only the numbers that are prime.

These calls were devised for the UNIX operating system and are not part of the ANSI C spec.

```
creat() Create a file for reading or writing
open() Opens a file for reading or writing
close() Close a file after reading or writing
unlink () Delete a file
write () Write bytes to file
read () Read bytes from file
```

Use of these system calls requires a header file named "fcntl.h":

```
# include <fcntl.h>
```

- The program takes the two file names first.out and second.out as arguments from the command line
- The program then checks if the second.out file is present. If not present then the second.out file is generated
- As files stores string type of data. So the strings from the first.out file are read and are converted in to integers by using sscanf() function
- The two numbers are separately recognized by a new line between them.
- The program then checks if a number is prime using the function called check_prime( )
- The prime numbers are then written in the file named second.dat
- The two files are closed then.
- It is necessary to input the desired parameters, the program fails to execute otherwise

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<stdbool.h> //for using bool type variables
#include<fcntl.h> //for using the system calls e.g., open(), close(), unlink(), read(), write()

bool check_prime(int input)
{
          bool flag = 0;
          int remainder = 0, i;

          for(i=2; i<input; i++)
          {
                remainder = input % i;

                if(remainder == 0)
                {
                       flag = 0;
                       break;
                }//ending if(remainder ==0)
                else
                {
                       flag = 1;
                       continue;
                }//ending else
          }//ending for(int i=1;)

          return(flag);

}//ending check_prime()
```

```
void main (int argc, char *argv[])
{
          bool check_prime(int);

          if (argc == 3)
          {
               struct stat statbuf;
               stat(argv[1], &statbuf);
               int filesize=statbuf.st_size;

               int fd1,fd2,check,i,input;
               bool result;
               char buf[filesize];


               fd1 = open(argv[1],O_RDONLY);
               fd2 = open(argv[2],O_RDWR|O_CREAT,S_IRWXU|S_IRWXG|S_IRWXO);

               do
               {
                       i = 0;
                               do
                               {
                                       check=read(fd1, buf+i, 1);
                                       i++;

                               }//ending do while ()
                               while(check != 0 && buf[i-1] != '\n');

                       buf[i-1] = '\0';
                       sscanf(buf, "%d", &input);

                       result = check_prime(input);

                       i=0;
                               if (result == 1)
                               {
                                       do
                                       {
                                               write(fd2,buf+i,1);
                                               i++;
                                       }//ending do while ()
                                       while(buf[i] != '\0');

                                       write(fd2,"\n",1);

                               }//ending if(result == 1)

               }//ending do while()
               while(check != 0);

               close(fd1);
               close(fd2);

          }//ending if (argc == 3)
          else
               printf("\n !Entered Paramenters Are Not Correct! \n");


}//ending main (int)
```

Exercise 2

Write a simplified version of the “ls” command receiving as a command line parameter the name of a directory and that visualizes the names of all the files contained in this directory.

- The header file <dirent.h> defines the DIR data type through typedef
- DIR is a type representing a directory stream
- It also defines the structure dirent
- The program takes the absolute path name from the command line
- struct dirent *dir is created used to store the path of the directory
- The loop then reads the contents of the provided directory and prints them on the secreen
- The loop runs until all the contents of the directory are read and displayed
- It is mandatory to input the desired parameters, the program fails to execute otherwise

```
#include<stdio.h>
#include<dirent.h>
//The <dirent.h> header defines the DIR data type through typedef
//DIR is a type representing a directory stream.
//It also defines the structure dirent

void main(int argc, char *argv[])
{
        if (argc == 2)
        {
                DIR *directory;
                struct dirent *dir;

                directory = opendir(argv[1]);

                while (dir = readdir(directory))
                {
                        printf("%s\n",dir->d_name);
                }//ending while()
        }//ending if()

        else
        {
                printf("\n Incorrect Parameters, write the absolute path! \n");
        }//ending else

}//ending main ()
```

Exercise 3

Write a program able to copy data contained in a file whose name is a command line parameter in a second file whose name is passed as a second parameter.

- The program takes the two input file name e.g. first.out and second.out
- The first file is opened, read and the copied into the second opened file
- The first file is copied until the program gets a negative value indicating the end of file and all the contents of the file are copied

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>


void main(int argc, char *argv[])
{
        if (argc == 3)
        {
                int fd1,fd2,LineRead,x;
                struct stat statbuf;

                stat(argv[1], &statbuf);
                x=statbuf.st_size;

                char buf[x];

                fd1 = open(argv[1], O_RDONLY);
                fd2 = open(argv[2], O_CREAT | O_WRONLY, S_IRWXU|S_IRWXG|S_IRWXO);

                while ((LineRead = read(fd1, buf, x)) > 0) //it will read line by line so if no line
is present numRead value will be less than 1
                {
                        write(fd2, buf, LineRead);
                }//ending while()
        }//ending if()

        else
        {
                printf("Number of parameters inserted is not correct\n");
        }//ending else
}//ending main ()
```

Exercise 4(a)

Write a simple program able to do a fork() and then to print messages both from the father and the son processes. Finally modify the program in order to have the messages printed from the father only after the son is finished.

- The program uses the fork function to create the processes
- Fork( ) returns a zero value to the pid (process ID) variable in case of child and returns a non-zero positive integer in case of parent
- The fork returns values to parent's and child's PID such that zero is passed to the child's PID and the PID of the child is returned to the parent

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)

{

  int pid;

  pid = fork(); //fork () returns value to pid
              //zero in case of child   OR
             //Positive integer greater than zero incase if parent

  if (pid==0)
  {
                              printf("\nCHILD HERE\n");

  }//ending if()
  else
  {
                              printf("\nPARENT HERE\n");

  }//ending else

  return 0;

}//ending main ()
```

Exercise 4(b)

Write a simple program able to do a fork ( ) and then to print messages both from the father and the son processes. Finally modify the program in order to have the messages printed from the father only after the son is finished.

```
#include <sys/types.h>  //wait() is defined in sys/types.h header file
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)

{

  int pid;

  pid = fork(); //The fork returns a value to pid
                //The Value zero is returned to the child's PID
                //The Process ID of the child is returned to the Parent's PID

  if (pid==0)
  {
          printf("\nCHILD HERE\n and i have finished my work!");
          exit(0);
  }//ending if()
  else
  {
          wait();
//The wait() function compells the parent to let it wait for the child to finish its process.
//Sometimes it happens that a parent goes off before its child ends. in that particular situation
//such child process become a semaphore.
//In such situation the child process without a parent is given under the control of init process.

          printf("\nPARENT HERE\n I waited for the child to finish its process.\n");
  }//ending else

  return 0;
}//edning main ()
```

Exercise 5

Write a C program able to run the following Unix commands:

I. cp /etc/passwd .
II. sort passwd –o mypasswd
III. cat mypasswd

- The program uses two PIDs in order to save the two different forks called two times in a program
- This makes three process i.e.; Father, Son, Grandson
- The passwd file in Red Hat linux is present at /ect/passwd, instead in Ubuntu it is present at /cp/etc/passwd. The process is done by grand son

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>

void main(int argc, char *argv[])
{
	int pid_ 1, pid_2;

	pid_1 = fork();

	//checks if the fork( ) fails
	//if fails prints the message on the screen
	//The following if( pid_1 < 0) executes the parent process
	if (pid_1 < 0)
	{
		printf("Fork Failed\n");

	}//ending if()

	//The following else executes the child process
	else if (pid_1 > 0)
	{
		wait();
		execlp("cat","cat","mypasswd",NULL);
	}//ending else

	//The following else executes the grand child process
	else
	{
		pid_2 = fork();

		if (pid_2 > 0)
		{
			wait();
			execlp("sort","sort","passwd","-o","mypasswd",NULL);
		}//ending if()

		else
		{
			execlp("cp","cp","/etc/passwd",".",NULL); //in Red Hat Linux the passwd file is present at /etc/passwd
                                                      //whereas in Ubuntu passwd file is present at cp/etc/passwd
			exit(0);
		}//ending else

		exit(0);
	}//ending else
}//ending main ( )
```

Exercise 6

Create a function mydup2( ) able to give the same functionalities of the dup2( ), but written using only the dup( ).

The program uses the "ls ¦ wc-w" command to verify the use of mydup2() working written using only the dup ( ) function. The function mydup2(), the position of the newfd is closed to get free the slot. Finally dup () function runs till newfd is reached. The dup () function is not used further anymore. In the end all the file descriptors where dup() was not required were close.

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

int main(void)
{
        int fda[2],x,i;
        int mydup2(int,int);
        pipe(fda);

        switch (fork())//fork() is called and its returned value is checked by the switch
        {
                case -1:       //checks for the fork () if failed?
                        printf("Fork Failed\n");
                break;

                case 0:      //child process is executed
                        i=mydup2(fda[1],1);
                        printf("I am child, Return of mydup2 %d\n",i);
                        close(fda[1]);
                        close(fda[0]);
                        execlp("ls","ls",(char *)0);
                break;

                default:  //parent process is executed
                        i=mydup2(fda[0],0);
                        printf("I am parent, Return of mydup2 %d\n",i);
                        close(fda[0]);
                        close(fda[1]);
                        execlp("wc","wc","-w",(char *)0);
                        wait();
                break;
        }//ending switch()
        return 0;
}//ending main()

int mydup2(int fda, int for_closing)
{
        int i, r;

        close(for_closing);

        int x[100],j=0;

        while((x[j]=dup(fda)) != for_closing && j < 100)
        {
                        j++;
        }//ending while()

        for(r=0;r<=j-1;r++)
        {
                close(x[r]);
        }//ending for()

        return x[j];
}//ending mydup2()
```

Exercise 7

Write a program who creates a pipe and after the fork ( ) uses the pipe to send a large file from one process to the other during the execution.

- The program inputs the file from the command line in order to calculate its size
- pipe is created and fork () is called to create the parent child processes
- Parent process closes the read of the pipe
- Child closes the write of the pipe
- The child is put to wait () on reading till the parent finishes writing files in the pipe
- The child displays the number of bytes read from reading the data in the pipe

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>

int main(int argc, char *argv[])
{
              int data_processed, file_pipes[2], fd1, file_read, numread, finalread=0, x,
fork_result, wrote;
              struct stat statbuf;

              stat(argv[1], &statbuf);
              printf("Size of file: %lu\n", statbuf.st_size);

              x=statbuf.st_size;

    char buffer[x];
    char buffer2[x];

    fd1 = open(argv[1],O_RDONLY);

    if (pipe(file_pipes) == 0)
    {
        fork_result = fork();

               if (fork_result==-1) //checks for the fork () if failed?
               {
                      printf("Fork Failed\n");
               }//ending if()

              else if (fork_result == 0) //child process is called
               {
                      close(file_pipes[1]);
                      numread = read(file_pipes[0], buffer2, x);
                      printf("Child read %d bytes \n",numread);
                      exit(0);
               }//ending else if()
               else //parent process is called
               {
                      close(file_pipes[0]);
                      file_read = read(fd1, buffer, x);
                      wrote=write(file_pipes[1], buffer, file_read);
                      printf("Parent Wrote %d bytes\n",wrote);
                      close(fd1);
                      wait();
               }//ending else
    }//ending if ()
    return 0;
}//ending main ()
```

Exercise 8

Write a simple producer-consumer program, in which the queue between the two processes (father and son, or two brothers) is at most of 4 tokens, each token being a character. Use semaphores to protect the producer-consumer operations.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/shm.h>
#define SHMSIZE 4

void semaphore_init (int* sw)
{
	if (pipe(sw) == -1) //checks for the init () process if fails!
	{
		printf ("init() process error\n");
		exit (-1);
	}//ending if( )
}//ending void semaphore_init()

void semaphore_wait (int* sw)
{
	char buffer;
	if (read(sw[0],&buffer,1) != 1) {
		printf ("Buffer value is %c. Error in wait()\n", buffer);
		exit (-1);
	}//ending if( )
}//ending void semaphore_wait( )

void semaphore_signal (int* sw)
{
	if (write(sw[1],"X",1) != 1) {
		printf ("Error in signal()\n");
		exit (-1);
	}//ending if( )
}//ending semaphore_signal()

int main(void)
{
	int mutex[2],full[2],empty[2],buffer[2],sem_in[2],n,shmid,read,write;
	int fork_result,N=4,i;
	char *shmPtr;

	semaphore_init (mutex);
	semaphore_init (full);
	semaphore_init (empty);
	semaphore_init (sem_in);

	pipe(buffer);

	semaphore_signal(mutex);
	for( i=1 ; i<=4 ; i++)
	{
		semaphore_signal(empty);
	}//ending for()

	printf("Enter number of letters to read (max 4) = ");
	scanf("%d",&read);

	printf("Enter number of letters to write (max 4) = ");
	scanf("%d",&write);

	fork_result = fork();

	if (fork_result == 0)
	{
		shmid=shmget(2041, SHMSIZE, 0666 | IPC_CREAT);
		shmPtr=shmat(shmid, 0, 0);

		int y=1,out=0,r,q;
```

```
                char x;

                while(y<=read)
                        {
                                semaphore_wait(full);
                                semaphore_wait(mutex);

                                printf("\nI am consumer, mutex, out is %d, character read from shared memory is %c",out,shmPtr[out]);
                                printf("\n");
                                out=(out+1)%N;

                                semaphore_signal(mutex);
                                semaphore_signal(empty);

                        y++;
                        sleep(2);
                }//ending while()
        }//ending if( )
        else
        {
                int y=1,in=0,r,q;

                shmid=shmget(2041, SHMSIZE, 0666 | IPC_CREAT);
                shmPtr=shmat(shmid, 0, 0);

                while(y<=write)
                        {
                                semaphore_wait(empty);
                                semaphore_wait(mutex);

                                shmPtr[in]='a'+in;
                                printf("\nI am Producer, mutex, is in %d, character written in shared memory is %c",in,shmPtr[in]);
                                printf("\n");
                                in=(in+1)%N;

                                semaphore_signal(mutex);
                                semaphore_signal(full);

                y++;
                sleep(2);
                }//ending while( )
        }//ending else
        return 0;
}//ending main ( )
```